PIC 单片机应用设计经验与技巧

美国微芯公司(Microch{p Technology Inc.)开发的 CMOS 工艺 PIC 系列 8 位单片机(RISC 微控制器),特别是采用内置第二代 Flash 存储器(40 年存储寿命)的微控制器在快速应用方面具有独到之处。由于其易用性和高可靠性,该系列微控制器稳居 8 位单片机全球出货量之首。PIC 系列单片机具有指令集简洁、简单易学、速度高、功能强、功耗低、价格低廉、体积小巧、适用性好及抗干扰能力强等特点,大量应用于汽车电气控制、电机控制、工业控制仪表和仪表、通信、家电、玩具、低功耗的测控应用等领域,在国内越来越受到广大设计者的欢迎,微芯公司的单片机已经成为目前单片机世界的主流产品。

PIC 8 位单片机内已经包含运算器、存储器、A / D、PWM、输入和输出 I / O(灌电流可达 25mA)、通信等常用接口,自由灵活的定义功能可以适应不同的控制要求。而不必增加额外的 IC 芯片。这样电路结构很简单,开发周期将大为缩短。

PIC16 系列单片机属于 PIC8 位单片机的中级型产品,采用 14 位的 RISC 指令系统。笔者使用 PIC16F716 单片机设计了一个电动机保护器,在设计过程中遇到很多问题,通过多方查找资料以及向 Microchip 公司技术人员寻求支持,问题一一得到解决。现将部分问题记录如下,与大家一起探讨。

1 ICD2 作为程序烧写的使用

1.1 ICD2 简介

MPLAB ICD2 在线调试器是一款低价位的 PIC 开发工具。它利用 Flash 工艺芯片的程序区自读写功能来实现仿真器调试功能;使用的软件平台是 Microchip 的 MPLAB IDE(集成开发环境软件包),兼容 Windows NT、Windows 2000 和 Windlows XP 等操作系统。其通信接口方式可以是 USB(最高可达 2Mb / s)或 RS-232 串行接口方式;工作电压范围为 2. 0～5. 5V,可支持最低 2. 0V 的低压调试。

MPLABICD2 可以支持大部分 Flash 工艺的芯片。它不仅可以用作调试器,同时还可以作为开发型的烧写器使用。

1. 2 ICD2 作为烧写器时的配置

烧写芯片的方式有两种:普通烧写和在线烧写。在线烧写是适合大批量生产方式的烧写办法。使用在线烧写时通常用户都已经把芯片焊到了板上,此时就要求用户板上有预留的烧写接口。用户板上的接口是通过一条 6 芯的扁平电缆与 ICD2 主机上同样的接口一一对应连接的。图 1 显示了 MPLAB ICD2 与目标板上模块连接插座的互连状况。

ICD 连接插座有 6 个引脚,但只使用了其中的 5 个引脚,分别是 VDD(电源)、VSS(地)、VPP(编程电压)、PGC(同步时钟)和 PGD(数据)。

1．3 ICD2 作为烧写器时容易出现的问题及解决方法

尽管 MPLAG ICD2 与目标板的互连非常简单，但是一不小心就会出现问题，基本上每一个 PIC 的入门者都会碰到类似的问题。下面就一些常见问题作简要叙述。

如图 1 所示，在 VPP 与 VDD 之间通常要串接一个上拉电阻(通常约为 10kΩ)，这样 VPP 线可置为低电平来手动复位 PICmicro 单片机。但是对一般设计者来说，都是采用上电自动复位。如果在这里采用集成器件 DMP809，那么就会导致连接不上，程序没有办法烧入。

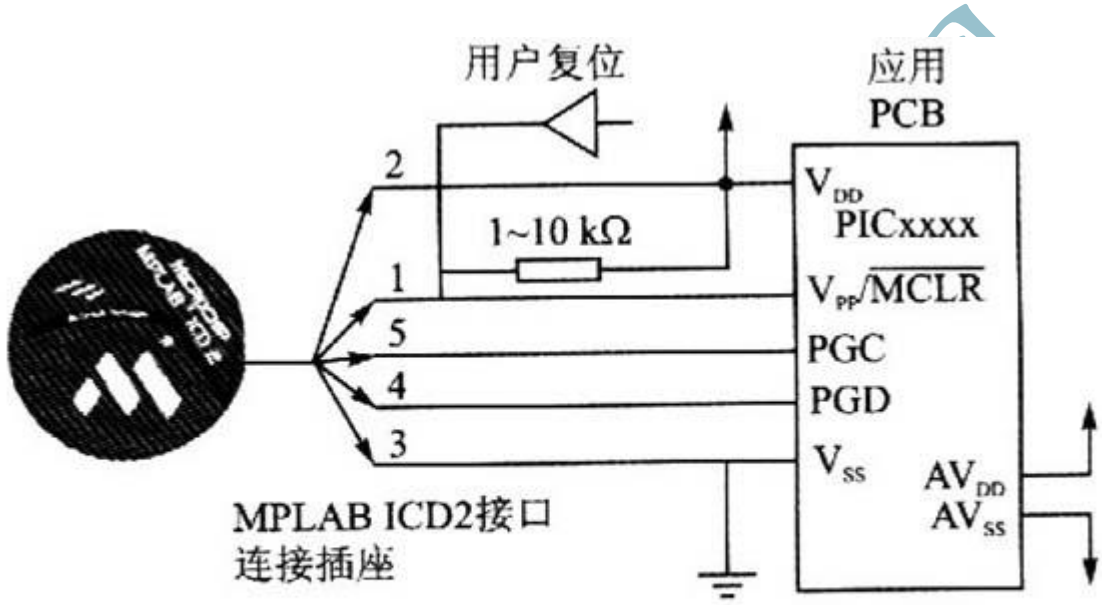

图 1　MPLAB ICD2 与目标板的连接

对于 PGC、PGD 两根线，由于在 ICD2 内部已经进行了上拉，所以在外围设计中，不要再进行上拉，否则会造成分压。对于 PGC、PGD 和 VPP 三根线，不要对地接电容，因为电容会阻碍在数据和时钟线上电平的快速转换，从而影响 ICD2 与目标板的连接。同样对于 PGC、PGD，由于数据或时钟都是双向传输的，这时如果在中间串一个二极管，则会影响 ICD2 与单片机的双向通信。

但是，对 PGC 和 PGD 来说，在单片机上同时复用为普通 I／O 口，而有些使用上必须要接对地电容或者是串接二极管。对于这种情况，唯一的处理方式就是在烧写时从芯片的 PGC 和 PGD 端口直接跳线到程序烧写口。

2 A／D 转换通道切换问题

笔者所设计的电动机保护器需要进行很多 A／D 转换，比如三相电流转换、零序电流转换以及各种*等。但是笔者所采用的 PIC16F716 单片机只有 5 路 A／D 转换通道，因此附加了一个多位选择开关对一个 A／D 通道进行复用。而在调试

中发现这样一个问题，就是 A / D 转换值不准确，甚至有点乱，但从程序流程以及代码角度均查不出任何问题。后查明 PIC16F716 单片机进行 A / D 转换通道切换时，需要一定的延时，延时时间是毫秒级。解决办法是：在通道间切换时，当第一个通道转换完成后，先转到另一个通道；然后延时 1ms 左右，再进行 A / D 转换。而对同一个通道信号切换时，要在第一个信号转换完成后，禁止信号输入，延时 1ms 左右；然后输人信号，再进行 A / D 转换。

这种做法比较麻烦，也很占用时间，并且从调试结果来看，问题并没有解决。在反复进行调试中，最后得到的优化解决办法是：对于通道间转换以及同一通道信号转换，要对每一个信号至少进行两次 A / D 转换；第一次的转换结果，舍弃不予处理，只取第二次 A / D 转换的结果。从调试结果来看，很好地解决了这一问题。

3 软件开发小技巧

PIC 单片机采用精简指令集，例如对于 PIC16F716 单片机，只有 35 条单字节指令。要用这么少的指令实现复杂的控制或计算，显然要在软件设计上多下功夫，并且 PIC 的指令系统与 51 系列单片机有很大不同，这让 PIC 初学者很不适应。下面笔者就自己的体会，谈一些软件设计需要注意的问题。

3.1 指令的大小写问题

编写 PIC 单片机的源程序，除了源程序的开始处需要严格的列表指令外，还须注意源程序中字母符号的大小写规则，否则在 PC 机上汇编程序时不会成功。在源程序中都会使用伪指令 INCLUDE。这条指令将列表中指定的单片机文件(在 MPLAB 中)读入源程序作为源程序的一部分，所以凡是 MPLAB 中有关该单片机已有的寄存器在源程序中无需再用赋值指令(EQU)赋值，这就使所建立的源程序大为简化。

此外，由于有了伪指令 INCLUDE，所以根据 MPLAB 软件中的格式，在源程序中的操作数凡是涉及 MPLAB 已规定的寄存器名称的，其字母一律只能大写，不能小写。其余操作码、符号字母可任意大小写，但 0x 中的 x 应小写。否则汇编不会成功。鉴于上述原因，为了书写方便，在使用 MPLAB 软件时，PIC 单片机的源程序均用大写字母为宜(0x 例外)。

3．2 振荡器的配置以及时序的计算

PIC 系列单片机可以工作于以下 4 种不同的振荡器方式：LP(低功耗晶体振荡器)、XT(晶体谐振器)、HS(高速晶体谐振器)和 RC(阻容振荡器)。用户可以根据其系统设计的需要，通过对配置位(FOSC1 和 FOSC2)编程，选择其中一种工作模式。

而一旦振荡器配置完成，那么根据用户的配置，可以轻松地计算出程序运行的时间以及 A / D 转换所占用的时间，这样就会很轻松地安排好单片机的时序。例如，如果采用 4 MHz 的 HS 振荡模式，那么单片机的时钟频率为 FOSC/4，也就

是说执行一条指令需要 1μ s；对于需要两个指令周期的指令，需要 2μ s。而对于 A / D 转换，如果 A / D 转换时钟位选择为 FOSC / 8，那么 A / D 转换模块转换一个位的时间 Tad 就为 2μ s。对一个 8 位的转换来说，需要的时间为 9. 5Tad，也就是完成一次 A / D 转换的时间为 19μ s。这样只需要查看源程序的行数并作简要分析，就可以计算出程序运行的时间。

3.3 存储体的选择

PIC 单片机的数据存储器通常分为两个存储体，即存储体 0(Bank0)和存储体 1(Bank1)。每个存储体都是由专用寄存器和通用寄存器两部分组成的。两个存储体中的一毡寄存器单元实际上是同一个寄存器单元，却又具有不同的地址。

不同型号的 PIC 单片机，其数据存储器的组成(即功能)是不完全相同的，所以设计人员一旦选用了某个 PIC 单片机的型号后，就要查找该单片机的数据存储器资料，以便编程使用。

笔者所采用的 PIC16F716 单片机的存储区，是通过 STATUS 寄存器的 RP1 位和 RP0 位来选择的。当配置为 00 时，表示选择存储区 0；当配置为 01 时，表示选择存储区 1。因为存储区的改变只须改变 RP0 位，所以通常在程序编写时，只改变 RP0 位来选择存储区。但是这样容易造成程序的混乱，因此，笔者建议在每次更换存储区时，要分别对 RP0 和 RP1 进行置位。在程序初始化时，最好将寄存器的初始化分为两部分：第一部分为存储区 0；第二部分为存储区 1。然后将每个需要初始化的寄存器分别在对应的存储区进行初始化即可。

3.4 GOTO 和 CALL 指令的不同使用

在 PIC 的汇编程序中，CALL 与 GOTO 指令使用的场台不同。CALL 是用来调用子程序的，在调用完子程序后返回到调用前的程序；而 GOTO 是无条件转移，即由此状态进入另外一个状态而不需要返回。

为了使程序更加具有可读性，使流程更加清晰、合理，通常程序都采用模块化程序设计，即将程序按照功能分成不同的子程序，而主程序则相当简洁，只须采用 CALL 指令对子程序进行调用。

由于 PIC 单片机的堆栈有限，在程序中不能无止境地使用 GOTO 指令，否则会使堆栈溢出，程序无法正常运行。但是在有些时候，例如当程序出现分支时，则不得不使用 GOTO 指令。对于 PIC16F7x 系列单片机，程序出现分支时只能通过 STATUS 寄存器的 Z 位或 C 位进行判断。这时在两种情况的前一种情况下，必须使用 GOTO 指令进行转移；否则在执行完第一种情况后，紧接着又执行第二种情况。程序如下：

```
BTFSS STATUS, Z

GOTO A
```

GOTO B

在跳转到 A 时，必须使用 GOTO 指令；否则执行完这条语句以后，紧接着执行 GOTO B。这样无论 Z 为何值，程序都将跳转到 B。而对于 GOTO B，则可以不必使用 GOTO 指令。

在上面这种情况下，由于 GOTO 只在子程序内部进行跳转，小程序内部循环占用堆栈的级数不多，因此使用 GOTO 指令是可行的。但是在大的程序中使用 GOTO 指令，将有可能无法返回到调用前的下一条指令。

因此，笔者建议，在使用汇编语言进行程序设计时，应该将程序分解成一级级的子程序；然后在程序之间进行调用，尽量将 GOTO 指令跳转的范围缩小。

3.5 对芯片的重复烧写

对没有硬件仿真器的设计者来说，总是选用带有 EPROM 的芯片来调试程序，通过反复的修改来观看运行结果，以便对程序进行调试。每更改一次程序．都是将原来的内容先擦除，再编程，浪费了相当多的时间，又缩短了芯片的使用寿命。如果后一次编程较前一次，仅是对应的机器码字节的相同位由 1 变为 0，那么就可在前一次编程芯片上再次写入数据，而不必擦除原片内容。

在程序调试

过程中，经常遇到常数的调整。如果常数的改变能保证对应位由 1 变 0，则都可在原片内容的基础上继续编程。另外，由于指令 NOP 对应的机器码为 00，调试过程中指令的删除，可先用 NOP 指令替代，编译后也可在原片内容上继续编程。

结语

在采用 PIC 单片机进行设计过程中，注意到 PIC 单片机自身的特点，可尽量少走弯路，从而缩短开发周期。同样在软件设计上采用合适的方法，可以使整个程序运行稳定，而且程序空间的使用也将有所减少，避免了调试中的 Bug。以上只是笔者在实际设计过程中一些小小的体会。希望与大家一起探讨，并在共同学习中为 PIC 单片机的普及和推广做出贡献。